Politecnico di Torino

Porto Institutional Repository

[Proceeding] Microprocessor fault-tolerance via on-the-fly partial reconfiguration

Original Citation:
Di Carlo S., Miele A., Prinetto P., Trapanese A. (2010). *Microprocessor fault-tolerance via on-the-fly partial reconfiguration.* In: IEEE 15th European Test Symposium (ETS), Praga, CZ, 24-28 May 2010. pp. 201-206

Availability:
This version is available at : http://porto.polito.it/2380367/ since: January 2011

Publisher:
IEEE Computer Society

Published version:
DOI:10.1109/ETSYM.2010.5512759

Terms of use:
This article is made available under terms and conditions applicable to Open Access Policy Article ("Public - All rights reserved") , as described at http://porto.polito.it/terms_and_conditions.html

Porto, the institutional repository of the Politecnico di Torino, is provided by the University Library and the IT-Services. The aim is to enable open access to all the world. Please share with us how this access benefits you. Your story matters.

(Article begins on next page)

Politecnico di Torino

Microprocessor fault-tolerance via on-the-fly partial reconfiguration

Authors: Di Carlo S., Miele A., Prinetto P., Trapanese A.,

Published in the Proceedings of the IEEE 15th European Test Symposium (ETS), 24-28 May 2010, Praga, CZ.

N.B. This is a copy of the ACCEPTED version of the manuscript. The final PUBLISHED manuscript is available on IEEE Xplore®:

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5512759

DOI: 10.1109/ETSYM.2010.5512759

© 2000 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Microprocessor fault-tolerance via on-the-fly partial reconfiguration

Stefano Di Carlo, Andrea Miele, Paolo Prinetto, Antonio Trapanese
Politecnico di Torino
Dipartimento di Automatica e Informatica
I-10129, Torino, Italy
e-mail:{firstname.lastname@polito.it}

***Abstract*—This paper presents a novel approach to exploit FPGA dynamic partial reconfiguration to improve the fault tolerance of complex microprocessor-based systems, with no need to statically reserve area to host redundant components. The proposed method not only improves the survivability of the system by allowing the online replacement of defective key parts of the processor, but also provides performance graceful degradation by executing in software the tasks that were executed in hardware before a fault and the subsequent reconfiguration happened. The advantage of the proposed approach is that thanks to a hardware hypervisor, the CPU is totally unaware of the reconfiguration happening in real-time, and there's no dependancy on the CPU to perform it. As proof of concepts a design using this idea has been developed, using the LEON3 open-source processor, synthesized on a Virtex 4 FPGA.**

***Index Terms*—fault tolerance, partial reconfiguration, self-repair architectures, graceful degradation**

I. Introduction

Embedded systems are nowdays widely used for many safety-critical applications that impose very strict, often conflicting, requirements, including fault-tolerance, real-time data processing, and reduced use of resources and power.

Several solutions do exist to guarantee hardware fault tolerance depending on the requirements of the target design. Neverthless, choosing among them often requires compromises in terms of cost, performance, fault detection delay, etc. Moreover, most of these techniques have been originally developed to be applied on ASICs (Application Specific Integrated Circuit), while nowdays several custom embedded SoCs (System on Chip) are realized on FPGAs (Field Programmable Gate Array). Even if the proposed techniques still mantain their effectiveness, they are not optimized to take advantage of most recent FPGA technologies, and they do not consider the nature of FPGA-specific faults.

Although the symptoms of transient or permanent faults on FPGAs can be categorized in component and control faults as proposed in [1], their source can be different and more complex to discover than on traditional ASICs. In a FPGA design, a permanent fault does not necessarily identify a damaged circuit; it can be generated by a bit-flipping of the FPGA configuration memory. Even if a more general approach could be used, traditional fault detection, diagnosis, and recovery methods for SRAM-based FPGAs tend to be very defect-tolerance oriented and technology specific. In [2] the authors review many fault detection mechanisms for FPGAs, mostly aiming at ensuring defect tolerance, while for fault tolerance they advise that a system level or a board level solution would be more effective and easy to implement than a low-level one.

A key technology enabling a new and generalized fault-tolerance architecture for FPGA SoCs is the "glitch-free" Dynamic Partial Reconfiguration feature of modern FPGAs [17], [16], [15]. Such a feature allows time-sharing of a predefined reconfigurable area on the FPGA fabric between multiple hardware instances. This feature has been used with success in some experimental designs ([3], [4]), but its full potential on the dependability side is yet to be exploited.

The present paper proposes a method to improve microprocessors fault tolerance through a self-repair strategy enabled by dynamic partial reconfiguration. The general idea is that when a pipeline stage is found to be defective, it is on-the-fly replaced by a new one, mapped into a reconfigurable area. In this way the spare unit is placed on the chip on demand, while its area can be used by other components during normal operation.

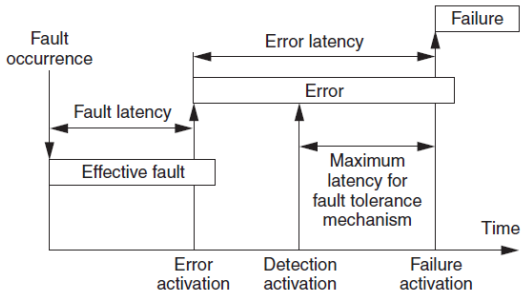

Figure 1. Error detection timing (©IEEE 2001, from [1])

Using as reference the taxonomy introduced in [5] and depicted in Figure 1, the proposed architecture aims at detecting an error and at correcting the fault that generated it through the means of reconfiguration before the error propagates and causes a failure. Typically the time between the Detection Activation and Failure Activation is too small to accomodate a device reconfiguration, thus a "freezing" technique is used to

Manuscript ACCEPTED version

delay the failure activation as much as needed to perform the required operations. In this way to resume the execution, there is no need for complex "rollback" or "restoration" operations.

The paper is organized as follows: in Section 2 we present the general concepts behind our proposed architecture; Section 3 presents a case study with related experimental results, while in Section 4 we draw some conclusions and outline some possible extensions and future work.

II. Proposed Architecture

The fault tolerant architecture presented in this paper comprises a set of "Replaceable Functional Units" (RFU), a set of "Spare Functional Units" (SFU), a Reconfiguration Manager and a Reconfigurable Area. Based on health status monitoring and other parameters, the Reconfiguration Manager decides if to use a SFU instead of the corresponding Replaceable Functional Unit.

Replaceable Functional Units can be critical or non critical, and corresponding Spare Functional Units can be hardware or software: critical RFUs are equipped with a concurrent error detection system and can be replaced by a hardware spare unit, while non critical RFUs may have or not a BIST facility and can be replaced by a software SFU.

Critical RFUs are initially hardwired on the device (i.e. they are placed in a non reconfigurable area), and when found defective, they are replaced by a SFU mapped inside the Reconfigurable Area. Non critical RFUs instead are initially mapped inside the Reconfigurable Area, and are replaced by a software SFU if the Reconfigurable Area is needed to host a critical spare unit, as depicted in Figure 2. The figure shows a critical Functional Unit (FU1) and a non critical Functional Unit (FU2); when FU1 fails, the corresponding SFU is placed inside the Reconfigurable Area, erasing the previously allocated FU2, which will be executed in software.

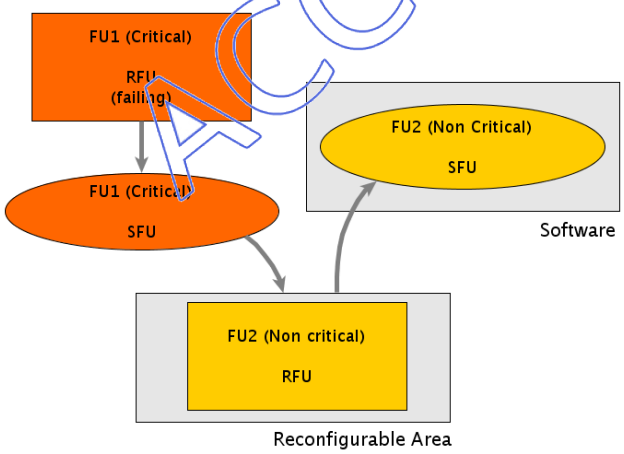

Figure 2. General Functional Units swapping architecture

The Reconfiguration Manager holds a repository of Spare Functional Units (in the main system memory, or a mass storage device), corresponding to an archive of device configuration bitstreams (for hardware SFU) and an archive of compiled modules (for software SFU).

The process of swapping a RFU with a SFU happens with the system being totally unaware of it, since the Reconfiguration Manager has the ability to "freeze" the involved circuits while the reconfiguration takes place. The "freezing" technique can be used by the Reconfiguration Manager to discriminate transient faults from permanent ones, thus avoiding unnecessary reconfigurations and transient error propagation at the same time.

The system awareness about current system configuration is granted through the "Reconfigurable Area Status Register", which can be read by other modules to perform operations coherently with the available hardware resources.

Though this approach is general and can be applied to a wide range of designs, in this paper we focus on a real-world scenario: a complex SoC synthesized on a FPGA, whose block diagram is depicted in Figure 3.

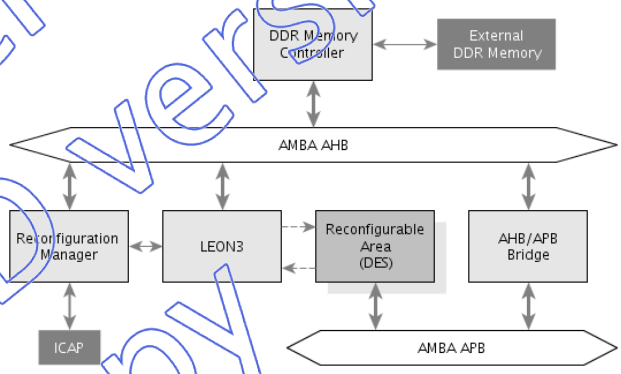

Figure 3. Conceptual block diagram of the proposed architecture

Such architecture addresses the error detection and recovery of the combinational logic of a processor's pipeline stage. It requires a reconfigurable area which is initially preallocated to a device of the SoC, a properly adapted processor, and a Reconfiguration Manager in charge of managing the resource allocation of the reconfigurable area. Once an error is detected, the CPU pipeline is "freezed", in order to keep all the combinational logic inputs constant, and the recovery process starts. This process consists in waiting few clock cycles before starting the reconfiguration, thus ensuring that the error is caused by a permanent fault. In case of transient fault, once the logic starts to behave normally, the error signal is immediately de-asserted and the execution resumes normally on the next clock cycle. The conceptual recovery process is described in Algorithm 1.

A. Reconfigurable Area

Dimensioning, positioning and connecting the reconfigurable area to the rest of the SoC are left to the designer, and depend on the particular modules (normal and spare) that should allocated on it. Once the reconfigurable area position is locked, the fixed interfaces need to be properly placed,

Manuscript ACCEPTED version copy

Algorithm 1 General description of the recovery process

```
if error detected {
   freeze the pipeline
   wait for N clock cycles
   if error still present {
      reconfigure with spare units
   }
   unfreeze the pipeline
}
```

keeping into account the maximum path length optimization, both inside and outside the reconfigurable module. Since the reconfigurable area size imposes an upper limit to the number of boundary-crossing nets (for example, "bus macros" on Xilinx FPGAs [16]), it may be necessary to use multiplexers and demultiplexers to use the same nets for multiple purposes.

B. Processor

In general to allow the online replacement of a part of the combinatorial logic of a CPU's pipeline stage, some modifications are required to the CPU architecture. As an example in this paper we aim at tolerating transient and permanent faults on the ALU, which is typically involved in the "Execute" stage of the pipeline.

Online concurrent error detection can be accomplished by using error detection codes, such as parity, on the ALU inputs, and an efficient concurrent code prediction logic, as proposed in [6]. Since the design of code prediction logic can be a complex and time-consuming task, a more practical approach can be used, as proposed in [7]. At high level a second unit is added, with outputs directly connected to the parity calculator. It is left to the synthesis tool to optimize the resulting logic, which will have a significantly lower area occupation than a second module, as reported in [7].

The error signal must be carried out of the processor to inform the reconfiguration manager of the problem, and has to be connected to the pipeline freezing logic. Once an error is detected all pipeline registers must be disabled in order to retain the previously stored data, avoiding error propagation. As illustrated above, this allows automatic tolerance of transient faults: if the error signal is deasserted within the few clock cycles that are counted between the error detection and the reconfiguration initiation, then the computation will resume without any loss or costly rollback operation.

In the same way, if the fault is permanent, the computation will resume after the time needed by the reconfiguration, still without any extra rollback operation.

In addition, the top level CPU interface must contain all ALU inputs and outputs, which need to be properly connected to the Execute stage signals and to the reconfigurable area interface, a multiplexer will select between the output coming from the internal ALU and the one coming from the external "spare" ALU. The ALU input signals will be simply connected to the reconfigurable module interface.

C. Reconfiguration Manager

The Reconfiguration Manager is the component responsible for the management of the reconfigurable area. In particular it is responsible of the following tasks:

- access the storage memory containing all reconfigurable modules bitstreams;
- access the configuration port of the reconfigurable area;
- monitor the CPU health status and manage the sequence of reconfiguration steps;
- mantain and export the information about the currently available device in the reconfigurable area.

In order to read the storage memory (which could be the main system memory, or an external slower memory, such as a flash memory), the Reconfiguration Manager has to be a bus master. In the particular case we implemented it as an AMBA AHB master. In addition it has to mantain a table with all bitstreams address mapped into the storage memory.

Which storage memory to choose depends on many factors, including reconfiguration time and thus the system availability in case of fault. An exhaustive exploration of different reconfiguration architectures for Xilinx FPGAs has been treated in [8], comparing the original Xilinx slave interface (which connects to the Xilinx OPB bus and is dependent on the CPU to operate) with some master interfaces able to read the main system memory and a totally SRAM-based device (in which the bitstream is fully contained inside the FPGA embedded SRAM memory blocks). The results presented in [8] show that obviously the totally SRAM-based solution is several orders of magnitude faster than the slave interface, reaching the physical limits of the Xilinx ICAP (Internal Configuration Access Port) interface. Anyway, the results also show that a well-designed master interface, with a small SRAM buffer, shows a significant advantage over the slave solution, comparable to the totally SRAM-based solution.

Moreover, the Reconfiguration Manager is responsible for the monitoring of CPU error signal, starting a reconfiguration once a fault has been detected as permanent, and has to deassert the "freeze" signal once the reconfiguration has completed.

D. Software Support

In general the proposed solution allows a total processor unawareness about the reconfiguration. Anyway the software needs to be informed about which core is currently placed inside the reconfigurable area, in order to avoid communicating with a peripheral that is not present anymore in the system. This can be accomplished by reading the Reconfigurable Area Status Register, which holds the "signature" of the currently available core. If a core is not available, its function is performed by a software Spare Functional Unit. The switching concept is introduced in [9], and has been adapted to keep into account the fact that the system cannot rely on the CPU during the reconfiguration. To avoid any ambiguity, the presence check should be done by the driver at the beginning and at the end of the communication with the device.

Manuscript ACCEPTED version

III. Case Study and Experimental Results

As a proof of concepts, we implemented the architecture described in the previous section using a SoC based on the LEON3 CPU ([10], [11]) in order to tolerate the ALU faults by replacing a previously allocated DES (Data Encryption Standard) crypto-core. According to the taxonomy previously introduced, we identify the ALU as a critical RFU, while the DES crypto-core as a non-critical RFU. Both of them have the corresponding SFUs (hardware for the ALU, software for the DES core). We decided to use this example because of the self-contained and general nature of both components, which also have a comparable area occupation. The resulting design has been synthesized and tested on a Xilinx Virtex 4 FPGA [17].

A. Processor architecture adaptation

LEON3 is a highly configurable 32-bit processor core conforming to the SPARC V8 architecture. It is designed for embedded applications, combining high performance with low complexity and low power consumption.

The LEON CPU is widely used in the aerospace industry and its fault-tolerant version has been validated through strict radiation tests [12]. Anyway, the implemented fault-tolerance techniques address typical radiation-induced ASIC faults involving sequential logic, only. This choice was motivated by the fact that the probability of a SEU on the combinational logic propagating into registers on a clock edge is low as reported in [13]. This assumption becomes less significant if the circuit is synthesized on a FPGA. A SEU on the configuration memory may cause permanent faults on the "combinational logic" of the CPU. The only configuration of the fault-tolerant LEON architecture that provides greater protection against this kind of faults is the master-checker mode [12], [1] (using 2 CPUs), which has a 100% area overhead, thus its use is limited to applications extremely demanding in terms of error detection.

Its pipeline is composed of 7 stages (Fetch, Decode, Register Access, Execute, Memory, Exception, Write Back).

In order to keep the complexity low, thus fitting the FPGA available for the implementation, the LEON3 has been configured with a minimal setup (no hardware MUL/DIV unit and no cache).

Due to the particular description style used in Gaisler libraries [14], the whole processor pipeline is described in a behavioral way, with just 2 processes, one implementing the full pipeline stages functionality, and the other one implementing the sequential logic.

This particular style showed both advantages and disadvantages for the purposes of our design: on one side it allowed the easy implementation of the pipeline freezing logic, with a simple modification to the "sequential" process; on the other one, it required additional reverse engineering work on the "combinational" process in order to isolate the ALU/shifter logic from the other logic implemented in the Execute stage of the pipeline.

Once the functionality has been clearly identified, a new entity (the hardware SFU) with the minimal interface to provide the same functionality has been developed, which implements the following operations:

- Logic (AND, NAND, OR, NOR, XOR, XNOR)
- Arithmetic (ADD/SUB)
- Shift

This unit has been set to be mapped on the reconfigurable area in case of fault detection. Moreover, following the approach suggested in [7], another "sample" of this unit has been inserted together with parity calculators on the input and on the output in order to provide error detection, leaving to the synthesis tools the task of optimizing the resulting logic. This simple scheme has been used because of its straightforward implementation and acceptable area overhead. In a more sophisticated architecture a Berger code based solution could be adopted, as suggested in [1].

The last modification had the scope of driving "out" of the CPU all signals needed to properly connect the external ALU. Outgoing signals are simply driven by the original signals, even if the external ALU is not present, while the incoming signals are connected to a 2-way multiplexer (implemented as a simple "if" statement in the combinational circuit).

To implement the pipeline freezing logic, the "freeze" signal has been conceptually connected to the enable input of pipeline registers. Practically the sequential process has been modified to take into account that if the freeze signal is high, no action has to be done in that process.

The overall resulting architecture is showed in Figure 4, where we emphasized the key modifications that we introduced in the "Execute" stage.

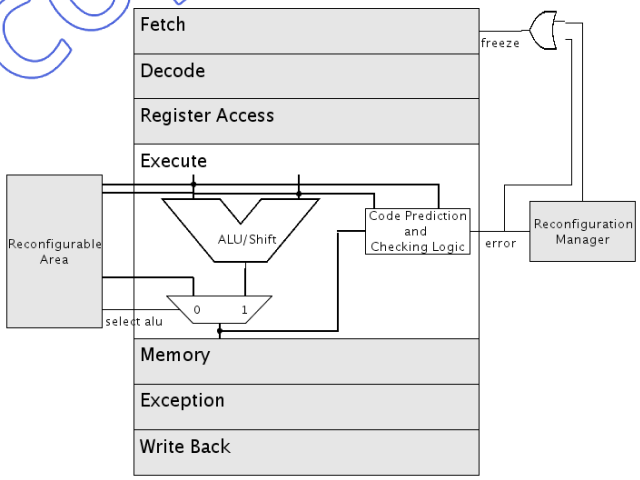

Figure 4. Detail of the modified LEON3 processor pipeline

B. Reconfiguration Manager Architecture

The Reconfiguration Manager is conceptually split into 3 different subsystems:

Manuscript Accepted Version Copy

- storage memory read and configuration memory write systems;
- processor pipeline freezing control;
- software support facilities.

The component is equipped with 2 different interfaces to the AMBA bus. At the same time it is an AHB master and an APB slave; the master interface is used to fetch the configuration bitstream from the storage memory (the main system memory or another external memory), while the slave interface is used to provide the software support facilities.

Access to the FPGA configuration memory is ensured through the 32-bit Virtex 4 ICAP port [17], which is instantiated as a black-box and mapped on the target device during place&routing. A buffer is used to store temporary bitstream data in order to achieve high performance, since this port can write up 32 bits per clock cycle [15].

Finally, the pipeline freezing control logic is responsible of both blocking the CPU operation during the reconfiguration and restoring it once it is completed. During normal operation the error signal is connected both to the Reconfiguration Manager and the pipeline "freeze" signal: as soon as the error signal is asserted, the pipeline is blocked and the Reconfiguration Manager starts an internal timer to check if the fault is permanent or transient. If at the end of the detection window the error is still present, the Reconfiguration Manager overrides the freeze signal to ensure that any glitch that may occur during reconfiguration doesn't propagate to the registers, and starts the reconfiguration.

C. Top Level Design and Reconfigurable Modules

Given the particular toolchain used for the synthesis on a Xilinx device [16], the top level design had to be modified by using bus macros to connect the static and reconfigurable part, which is instantiated as a black box. During place & routing the reconfigurable area size has been set, and the bus macros had to be statically placed on the boundary; during the placement we kept into account the effects that the placement of bus macros has on the resulting layouts of both the static and reconfigurable parts.

The RFUs/SFUs that need to be hosted inside the reconfigurable area need almost no modification compared to a normal design. The only requirement is that they must share the same entity interface declaration: the APB DES interface must contain the ALU interface signals, and the external ALU interface must contain the APB signals, even if these are left unconnected inside the entity.

In order to reduce the number of signals crossing the reconfigurable area boundary, the APB interface has been splitted in a static and a dynamic part: the static one contains fixed configuration data used for PnP (Plug and Play) device detection, while the dynamic part is used for actual data transfers, interrupt routing and device addressing, and is the only one that actually passes the reconfigurable area boundary through bus macros.

D. Experimental Results

The system has been synthesized on the Xilinx ML403 board, equipped with the XC4VFX12 FPGA [17] and several experiments have been executed, in order to validate the concepts introduced in previous sections. The final design occupied almost all of the logic resources available on the FPGA (as reported in Table I), and the operating frequency remained unchanged (66MHz) compared to the static reference design provided with Gaisler libraries for that board.

	Virtex 4 Slices
Device capacity	5472
Static	3426
Reconfigurable Area Capacity	536
DES	313
External ALU	242

Table I
FPGA RESOURCES ALLOCATION

The system has been tested with a bare C prototype program which executed a full DES encryption/decryption cycle; the program included all the facilities to measure the execution times and to dynamically use the software version if the hardware device is not present anymore. Monitoring execution times allowed us to quantify the performance degradation due to the use of software libraries instead of the hardware device, and to analyze the reconfiguration time impact on the whole execution time.

DES encryption (20KB plain text)	*Time*
Pre-Reconfig (Hardware)	183 ms
Post-Reconfig (Software)	1500 ms

Table II
ENCRYPTION PERFORMANCE BEFORE/AFTER RECONFIGURATION

Measurements showed that the system is able to keep working normally after a reconfiguration, since only the DES encryption/decryption time is affected. Table II shows one order of magnitude performance degradation due to the lack of hardware support). It is interesting to note that the resulting layout kept working at the original clock frequency, which has been set with a wide safety margin over critical path length.

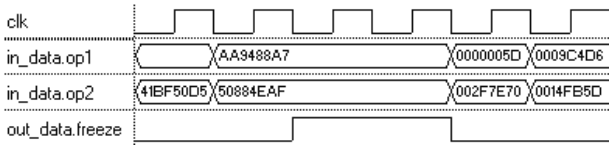

Figure 5. Simulation of the pipeline freezing logic (detail of ALU operands)

To simulate a fault, the error signal has been connected to one of the board switches instead of the error detection logic. This allowed us to cause a full reconfiguration cycle during the execution of the program. As the simulation results show in Figure 5, the freezing logic allows keeping constant the combinational circuits inputs during the reconfiguration. The

Manuscript ACCEPTED Version Copy

figure shows the detail of the clock and freeze signals, together with the ALU operands. Once the freeze signal is de-asserted, the computation resumes without any further operation.

Using the ICAP interface, reprogramming the reconfigurable area with the External ALU partial bitstream required on average 0,5 ms, which in terms of bandwidth is equivalent to about 150MB/s. This figure is acceptable for most systems, and scales almost linearly with clock frequency, as far as the memory throughput doesn't become a bottleneck.

Bitstream	*Size*
Full Initial (Static + DES)	581,8 KB
External ALU	67,2 KB
DES crypto-core	84,4 KB
Blanking bitstream	51,2 KB

Table III
BITSTREAM SIZES

Obiviously the actual reconfiguration time depends on the size of the partial bitstream (Table III), which becomes another important parameter from a system point of view: trying to minimize the "area" occupied by a component not only allows to improve device utilization, but could lead to a slightly faster reconfiguration time, as well.

IV. CONCLUSIONS AND FUTURE WORK

In this paper we introduced the concept of "fault-tolerance on demand"; thanks to the proposed architecture we have been able to verify that concept. We showed that it is possible to create a dependable system, which becomes "fault-tolerant" only when a fault is actually detected, thus limiting the compromises typically associated with the design of such systems.

By using the "freeze & resume" technique we showed that we can tolerate transient and permanent faults on a critical Replaceable Functional Unit by the means of FPGA dynamic partial reconfiguration. Moreover, we demonstrated that by using the appropriate techniques, the reconfiguration (which actually removes an IP core from the device) doesn't affect system functionality, but only causes performance degradation when executing some specific tasks.

The prototype presented in this paper can be extended with a more complex architecture, in order to cover the faults happening in any pipeline stage, and could be the starting point to evaluate more complex recovery techniques for FPGA-based designs.

REFERENCES

[1] M. Pflanz and H. T. Vierhaus, "Online check and recovery techniques for dependable embedded processors," *IEEE Micro*, vol. 21, pp. 24–40, Sept. 2001.

[2] A. Doumar and H. Ito, "Detecting, diagnosing, and tolerating faults in sram-based field programmable gate arrays: a survey," *IEEE Trans. VLSI Syst.*, vol. 11, pp. 386–405, June 2003.

[3] P. Sedcole, P. Y. K. Cheung, G. A. Constantinides, and W. Luk, "Run-time integration of reconfigurable video processing systems," *IEEE Trans. VLSI Syst.*, vol. 15, pp. 1003–1016, Sept. 2007.

[4] A. Tumeo, S. Borgio, D. Bosisio, M. Monchiero, G. Palermo, F. Ferrandi, and D. Sciuto, "A multiprocessor self-reconfigurable jpeg2000 encoder," in *Proc. IEEE International Symposium on Parallel. Distributed Processing IPDPS 2009*, pp. 1–8, May 23–29, 2009.

[5] J. A. Clark and D. K. Pradhan, "Fault injection: a method for validating computer-system dependability," *Computer*, vol. 28, pp. 47–56, June 1995.

[6] S.-B. Ko and J.-C. Lo, "Efficient realization of parity prediction functions in fpgas," *J. Electron. Test.*, vol. 20, no. 5, pp. 489–499, 2004.

[7] M. Portolan and R. Leveugle, "A highly flexible hardened rtl processor core based on leon," in *Proc. 8th European Conference on Radiation and Its Effects on Components and Systems RADECS 2005*, pp. J7–1–J7–6, Sept. 19–23, 2005.

[8] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time partial reconfiguration speed investigation and architectural design space exploration," in *Proc. International Conference on Field Programmable Logic and Applications FPL 2009*, pp. 498–502, Aug. 2009.

[9] A. S. Stefano Di Carlo, Paolo Prinetto, "A fpga-based reconfigurable software architecture for highly dependable systems," *ATS Conference Proceedings*, 2009.

[10] Gaisler Research, *GRLIB IP Library User's Manual*, v1.0.21 ed., 2009.

[11] Gaisler Research, *GRLIB IP Core User's Manuel*, 1.0.21 ed., 2009.

[12] J. Gaisler, "A portable and fault-tolerant microprocessor based on the sparc v8 architecture," in *Proc. International Conference on Dependable Systems and Networks DSN 2002*, pp. 409–415, June 23–26, 2002.

[13] P. Liden, P. Dahlgren, R. Johansson, and J. Karlsson, "On latching probability of particle induced transients in combinational networks," in *Proc. Twenty-Fourth International Symposium on Fault-Tolerant Computing FTCS-24. Digest of Papers*, pp. 340–349, June 15–17, 1994.

[14] J. Gaisler, *A structured VHDL design method*. Gaisler Research, http://www.gaisler.com/doc/vhdl2proc.pdf.

[15] Xilinx Inc., *Virtex-4 FPGA Configuration User Guide*, v1.11 ed., 2009.

[16] Xilinx Inc., *Early Access Partial Reconfiguration User Guide*, v1.2 ed., 2008.

[17] Xilinx Inc., *Virtex 4 Family Overview*, v3.0 ed., 2007.

Manuscript ACCEPTED Version Copy